

Programmable Shading for High Performance Computing

Mark A. Bolstad
Raytheon / Army Research Lab MSRC

Abstract

Programmable shading is a technique that allows for flexibility in specifying the appearance of an object independent from its underlying geometry. Widely used in feature films and by the entertainment industry for more than a decade, programmable shading has not been adopted by the scientific visualization community due to its lack of interactivity. With the advent of graphics hardware that supports programmable shading, and the two major graphics APIs (OpenGL and DirectX) moving to where programmable shading will be the standard method of defining appearances, these technologies will allow scientific visualization to embrace these technologies by enabling real time interaction.

Programmable shading can facilitate the exploration of data by allowing the end user to develop new methods for mapping the data, independent of the software used to create the geometry. This paper provides an overview of rendering in general, examples of programmable shading from entertainment, ARL MSRC, and other HPC sites, and shows how the recent advances in graphics hardware and associated Application Programming Interfaces (API) will enable the DoD user community to apply these technologies to visualization.

1 Introduction

Rendering is the process of converting a description of a three-dimensional scene consisting of geometry, appearance and lighting, into a two-dimensional image. Rendering consists of several steps: hidden-surface removal, illumination, and shading.

Hidden-surface removal (or visible-surface determination) is the process of determining what is currently visible from a viewpoint. Hidden-surface algorithms in use today are variants of point-sampling algorithms, where visibility is determined by taking sample points at a finite number of points, and then trying to infer the visibility of an entire surface from these limited points. Sutherland, et al. [Sutherland 1974], identified sorting order as the major distinction between different types of point-sampling algorithms. Three of the ten algorithms that Sutherland identified are the most commonly used today: Z-Buffer [Catmull 1974], Ray-Tracing [Appel 1968] [Kay 1979] [Whitted 1980], and Scan-Line [Wylie 1967] [Watkins 1970] [Bouknight 1970a] [Bouknight 1970b]. The Z-Buffer algorithm forms the core of most hardware-accelerated graphics processors.

Illumination is the process of determining the components of the scene that are dependent on the propagation of light through that scene. Illumination is divided into two major components, direct illumination, determining the contribution of light directly deposited on the surface of an object from a light source, and indirect illumination, where the light impinging on an object has arrived at the surface through one or more bounces from other surfaces. The process of calculating the contribution from a

direct illumination source is a variant of the hidden-surface process described above.

Shading is the process of determining the distribution of light leaving a surface given the incident light and the optical properties of that surface. Using the bi-directional reflectance distribution function (BRDF), shading computes the final color of the surface in the image. The BRDF or shading models implemented in hardware today are Gouraud [Gouraud 1971], Phong [Bui-Tuong 1975], and Blinn [Blinn 1977],

These shading models limit the appearance of objects to at most a handful of appearance types, and do not begin to describe the wide variety of materials seen in the physical world. To overcome these limitations, various other techniques were developed including texture mapping [Catmull 1974], Bump Mapping [Blinn 1978], and Reflection Mapping [Blinn 1976]. The problem with these shading models and their extensions is that the appearance is directly tied to the underlying geometry. In order to change the appearance of the surface, it is required to change the description of the appearance at the definition of the geometry.

In 1987, Cook et al. [Cook 1987] described a new rendering architecture called REYES. Developed at Lucasfilm, Ltd., and refined at Pixar, REYES was designed to handle large scene complexity (efficiently render scenes with more than 80 million polygons), scene diversity (many different types of primitives including polygons, Non-Uniform Rational B-Splines, and fractals), and shading complexity that could mimic materials seen in the physical world. Based on these requirements, the authors decided that a programmable shading model was the only technique that would suffice. The shading model for REYES was based on Shade Trees [Cook 1984] and described in detail by Hanrahan and Lawson [Hanrahan 1990].

Description

Programmable shading is a technique that allows for flexibility in specifying the appearance of an object independent from its underlying geometry. By separating shape from shading, programmable shading alleviates the problem of adding complexity to geometry in order to describe a shading problem. For example, whereas cloth can be described as a net of polygons or a parametric surface for each individual thread, another approach would be to describe the overall shape of the fabric geometrically, taking the appearance of the cloth as a shading problem.

A programmable shader is a piece of code, usually a procedure that is run whenever the renderer needs the appearance of an object at a particular sample. Since the shader is called at point samples, it integrates easily with the three hidden-surface algorithms described earlier.

Figure 1 shows an example RenderMan shader that implements a plastic appearance, the associated scene description, and the resulting image.

```

surface
plastic (float Ka = 1, Kd = 0.5,
        Ks = 0.5, roughness = 0.1;
        color specularcolor = 1;)
{
    normal Nf = faceforward (normalize(N),I);
    Ci = Cs * (Ka*ambient() + Kd*diffuse(Nf)) +
        specularcolor * Ks *
        specular(Nf,normalize(I),roughness);
    Oi = Os;
    Ci *= Oi;
}

Display "purple_ball.tif" "file" "rgb"
Format 256 256 1
Projection "perspective" "fov" 30
Translate 0 0 5
WorldBegin
TransformBegin
Translate 0 4 -4
LightSource "pointlight" 1 "intensity" 70
TransformEnd
Color [ 1 .6 1 ]
Surface "plastic" "Ks" 1
Sphere 0.5 -0.5 0.5 360
WorldEnd

```

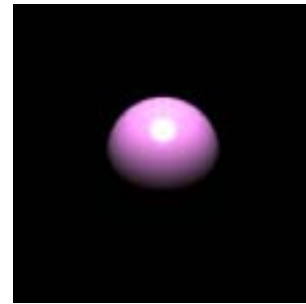


Figure 1. RenderMan "plastic" shader, a simple scene, and the associated image.

2 Programmable Shading in Entertainment

The entertainment industry has relied heavily on programmable shading from the initial development of the technique. First utilized by the feature film industry, programmable shading is now showing up in consumer games and in gaming consoles such as the PlayStation2 and the Xbox.

The earliest use of programmable shading was in a little known 1984 short, *The Adventures of Andre and Wally B.*, an animation created to test the capabilities of the REYES architecture. But the success of the 1986 Academy Award nominated short, *Luxo, Jr.*, and the 1989 feature film, *The Abyss*, showed the entertainment industry what programmable shading could achieve, and led the revolution of the inclusion of Computer Generated Imagery (CGI) into nearly every feature film produced since.

Since HPC is a different industry with different requirements from the entertainment industry, the question needs to be asked: *Why should HPC care about Hollywood?*

The first reason is with the advent of the all computer-generated movie, the film industry has had to deal with HPC-sized data management issues. The first Pixar/Disney movie, *Toy Story* (1995), required 1 Terabyte of data to describe the entire film, and 0.5 Terabytes to store the final generated imagery. For the film, *Shrek* (2001), the PDI/Dreamworks render farm consisted of 1482 processors with 6.5 Terabytes of storage. The entire film required around 200 Terabytes of storage for the description of the geometry, textures, and shaders used for each frame of the film.

The second reason is in the visual complexity of the scenes generated for modern film. Statistics from the movie *Shrek* indicate that each frame of the film contains from 200 million to over 1.2 billion polygons [Flarg 2001]. Additionally, each frame averages two gigabytes of input to render the image.

Is there any reason why HPC shouldn't adapt to all of the techniques the entertainment industry uses to make feature films? The main reason is that the goals of HPC and the entertainment industry are fundamentally different. For HPC, the parts of the physical world we are trying to model and visualize must stay as true to the object or principle that we are modeling to preserve reality. For the entertainment industry, the goal is to create a verisimilitude of the object or process that is to be depicted, using only as much of reality as necessary to

create the effect. For example, in the movie, *The Perfect Storm*, Industrial Light and Magic used models of ocean wave development and the optical properties of water as the principal components of their wave simulations, but had artists paint additional foam and spray elements to meet the look requested by the director. However, these differences are independent of the choice to use programmable shading, and its flexibility gives the user control over the depiction of reality.

3 Programmable Shading for High Performance Computing

Since programmable shading has been shown to handle scenes of high complexity and large data sets, how can we apply the techniques developed by the entertainment industry over the last decade to visualization and to DoD researchers supported by the HPC program? After understanding the mechanisms of programmable shading, it is a relatively simple process to add these features into an already existing visualization pipeline. The following set of examples will demonstrate the capabilities of using programmable shading with visualization.

The simplest use of programmable shading is to change the appearance from the standard Gouraud shading available in visualization software to a shader that will provide normal interpolation for curvature information (see Figure 2). By this simple transformation, DoD researchers have been able to acquire a better understanding of the shape of the surface, without obscuring the underlying visualization.

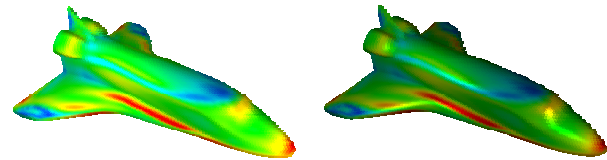


Figure 2. Gouraud (left) vs. Metal (right) shading.

For the Computational Fluid Dynamics simulation of the transport of species through a city block, a shader was designed that would create the illusion of volume rendering from a set of stacked iso-surfaces (Figure 3). For this image, iso-surfaces were extracted from the time varying, 4.5 million unstructured cell data as a pre-processing step, a later version of the shader

extracted the iso-surfaces from the data directly, thereby reducing both the disk and memory requirement of the renderer.

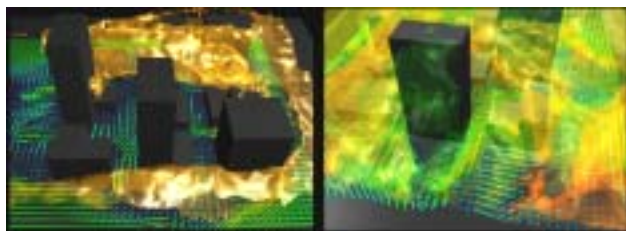


Figure 3. Species transport around a city block. "Gold Foil" (left) and "Whispy" (right) shaders.

Shown below (Figure 4) is a single frame from a 61,000-frame hologram of the Brilliant Anti-Armor Technology device.

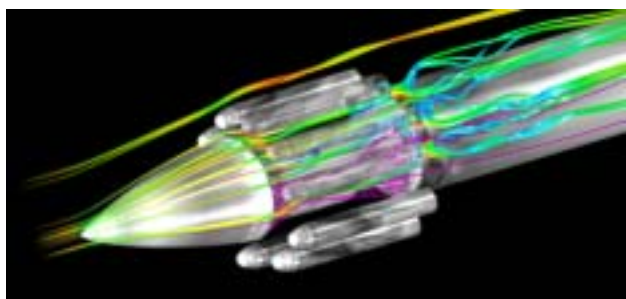


Figure 4. One frame from a hologram of the Brilliant Anti-Armor Technology device.

To prevent visual "tearing" of the image (caused when geometry is partially clipped by the edge of the image), a shader was created that would fade geometry, in this case, the streamlines extracted from the CFD simulation that was computed on the ARL MSRC resources, to transparent across a user-specified distance. Additionally, a shader that gave strong specular highlights was used to provide shape information, especially in the transparent sub-munitions.

3.1 Texture Synthesis

Another use for programmable shading is in synthesizing textures to convey additional information, such as, the curvature of a surface or multiple attributes. Figure 5 shows several examples of how generating textures can provide additional information on the shape of a transparent surface.

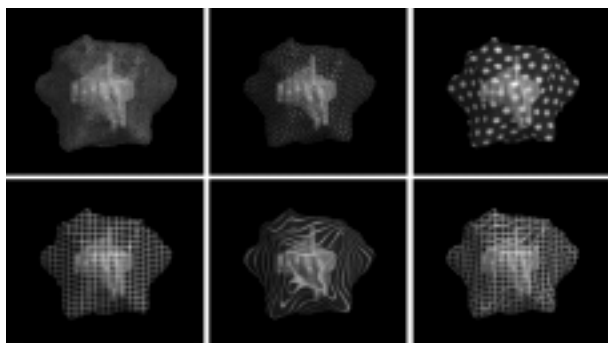


Figure 5. Examples of texture synthesis [Interrante 1997].

3.1.1 Textures for Surface Interpolation

Rheingans [Rheingans 1996] performed a study examining how using modulated textures instead of transparency helped computational chemists identify features on the solvent accessible surface of a molecule. This study showed that the chemists had better recognition of the surface shape, and a clearer understanding of the underlying model. While custom software was developed to create the textures, the technique could easily be implemented in a programmable shader (see Figure 6).

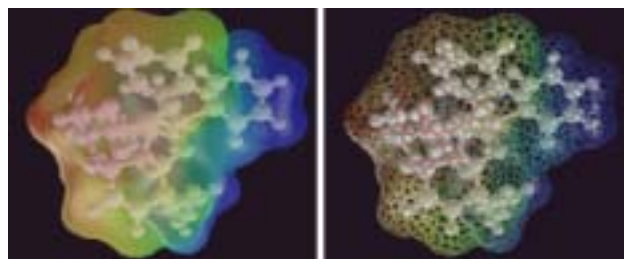


Figure 6. Comparison of transparency vs. texturing for surface curvature.

By encoding the curvature along principal directions as textured strokes and coloring by the magnitude of the gradient, Interrante [Interrante 1997] found that more information was conveyed to a viewer than by plain transparency alone (see Figure 7).

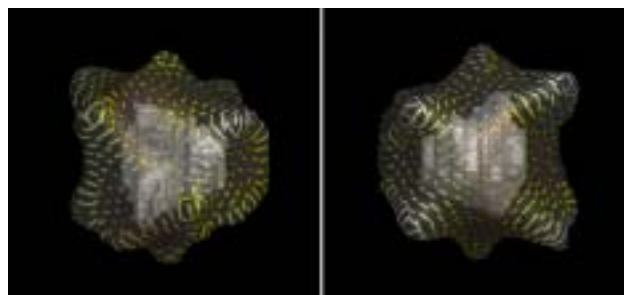


Figure 7. Texture synthesis showing components of principal directions, colored by the gradient of the curvature [Interrante 1997].

3.1.2 Textures for Encoding Data

Cabral and Leedom [Cabral 1993] developed a technique using Line Integral Convolution to create texture maps of two-dimensional vector fields. The fundamental concept is to render a scalar pixel value for each vector in a vector field. The pixel encodes the local vector field by convolving a set of input image pixels that lie under a local streamline. The resulting image or texture is similar to the input image, except it is blurred in the direction of the vector field. If the input image is uncorrelated white noise, streaks resembling fine hairs lie along streamlines in the vector field (See Figure 8).

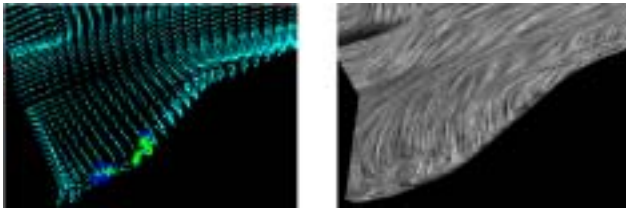


Figure 8. Line Integral Convolution of the surface vector field on the shuttle.

3.2 Volume Rendering

Unlike the techniques mentioned in previous sections, volume rendering is not a segmentation technique, i.e., it operates on the entire volume of data. Programmable shading languages usually do not have the capabilities or feature sets necessary to do volume rendering. However, there are two different methods that one can use to create volume rendered images inside a renderer supporting programmable shading.

3.2.1 Texture Mapping

The first technique for creating volume rendered images is to generate a series of tiled, gray-scale, texture maps that represent the data, and a separate texture map for the color and opacity transfer functions. Figure 9 shows an example set of input textures, and the resulting image for a continuous, spherical, density function. To render this dataset, a scene is created with a box at the extents of the original data, and a volume texturing shader is applied to each of the six faces.

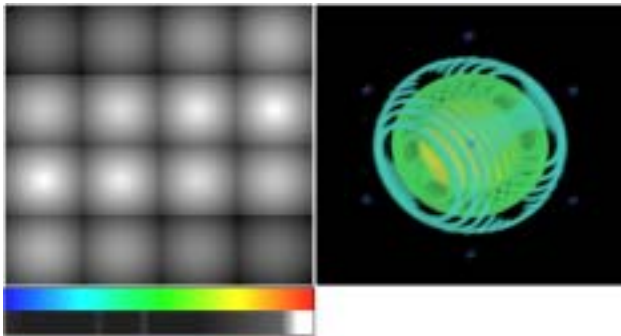


Figure 9. Texture Mapping for Volume Rendering. The images in the upper left are slices through the data set from front to back. The image in the lower left is the transfer map for color (top) and opacity (bottom) where black is transparent and white is fully opaque. The right image is the result.

The advantage of this technique is that you can embed geometry in the region covered by the volume, for example, the buildings in Figure 3, and have it rendered correctly. No other volume rendering technique is capable of mixing geometry types in this manner. The disadvantage of this technique is that your rendering is only as good as the number of image slices from the original data in the texture map.

Dave Bock at NCSA [Bock 1998] used this technique to create several animations of volumetric data. Figure 10 is one frame from the computation of the collision of binary neutron stars.

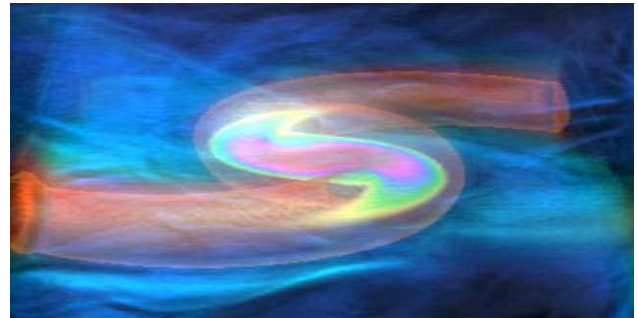


Figure 10. Volume rendering of the collision of binary neutron stars [Bock 1998].

3.2.2 Ray Casting

The second method is an application of a traditional ray-casting volume renderer built into a shader. This technique involves the same scene as in the previous example, but the name of the dataset is passed to the shader as opposed to an image. The shader, shown in Figure 11, generates a ray from the eye through the intersection point on the box. That ray, along with other information, is passed to an external C routine that computes the color and opacity along the ray. The shader then composites that color into the color returned from all other geometric objects in the scene. The external C routine used for Figure 11 was implemented in OpenDX, but could have been implemented in the Visualization Toolkit (VTK) as well.

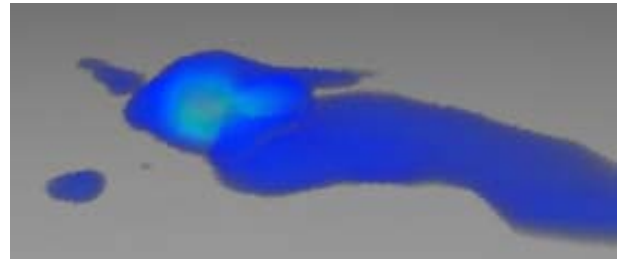


Figure 11. Direct volume rendering and the associated shader.

This technique cannot be implemented in the programmable graphics hardware available today. Purcell et al. [Purcell 2002] have hypothesized that with the expected modifications to existing hardware, ray casting will become a viable technique within the next year.

4 Programmable Graphics Hardware

Kurt Akeley [Akeley 2001] created a classification of "GL"-like machines based on the capabilities of graphics hardware (see Table 1).






	ERA	CAPABILITY	EXAMPLE
1	Pre-1987	Wireframe	
2	1987-1992	Shaded	
3	1992-2000	Texture Mapping	
4	2001-	Programmability - Programmable Shading	 NVIDIA
5	???	Global Evaluation - Ray Tracing - True Shadows - Global Lighting	 Henrik Wann Jensen

Table 1. Generational capabilities of GL-like hardware, after [Akeley 2001].

Peercy et al. [Peercy 2000] demonstrated that many of the capabilities of the RenderMan Shading language [Pixar 2000] (the de facto standard for programmable shading) could be implemented in 3rd generation hardware that supports multi-texturing extensions.

True 4th generation hardware is currently available in modern graphics chips, such as the NVIDIA GeForce4 [NVIDIA 2002] and the ATI Radeon 8500 [ATI 2001]. These chips replace the static functions for vertex and texture (fragment) operations with programmable ones. Since programmable hardware is in its infancy, there are many limitations, such as:

- Vertex and fragment programs have limited instructions.
- Programs cannot loop or conditionally branch.
- Each implementation has different instructions, capabilities, and evaluation sequences.
- Many resource limitations exist:

- Limited number of instructions;
- Limited number of registers;
- Limited inputs and outputs;
- Limited number of textures;
- Limited to 8-bits of fixed-point precision in fragment operations.

Programmable shading implementations [Stanford 2000] [SGI 2000] for 3rd generation hardware have none of the resource limitations described above, but suffer from other drawbacks:

- No true displacement of geometry
- Speed
 - The Stanford system compiles shaders on the fly before passing to the hardware.
 - Both systems require multiple passes through the graphics hardware to evaluate the shader.

In order to provide uniform access to programmable graphics hardware, several Application Programming Interfaces (APIs) have been created or proposed. These APIs target capabilities beyond what is currently available in today's hardware, but what will be implemented in future graphics chips.

5 APIs for Programmable Graphics Hardware

There are currently three publicly available APIs that define interfaces for programmable shading. The first and original specification is the RenderMan Shading Language. While somewhat outdated, and missing features available in modern programming languages (e.g., shaders tend to be monolithic since shaders may not call other shaders), the RenderMan Shading Language is still the most complete shading language currently in use. There are several software implementations of the RenderMan Standard [DOTC 2001], and there have been at least two attempts to build hardware implementations of the RenderMan standard, namely, the Pixar Vision Computer and the RenderDrive [ART 2002]. There have been reports of software to translate RenderMan shading language into OpenGL [Peercy 2000], but the software has not been made publicly available. All of the images created in Section 3 were made with a RenderMan-compliant software renderer.

The second API is the DirectX/Direct3D suite from Microsoft [DirectX 2001]. The DirectX API is capable of a wide range of programmable effects. Its development emphasis is directed at the games market, but much of the feature set could be leveraged for HPC applications. There are several features of the API that make it unsuitable for HPC type applications:

- Shaders are written in a pseudo-assembly language with little to no support for a high-level language API. However, shaders can be accessed from any high-level language.
- DirectX is a single platform API, limited to the Windows operating system.

The final API that will be discussed is the OpenGL 2.0 proposal, currently in development. The first efforts to define OpenGL began in 1990 and resulted in a specification in 1992. Over the last decade, the specification has changed very little while there has been a dramatic change in the capabilities of graphics

hardware. The OpenGL 2.0 proposal was motivated by the fact that [OpenGL2 2002]:

“... graphics hardware is changing rapidly from the model of a fixed function state machine (as originally targeted by OpenGL) to that of a highly flexible and programmable machine.”

With this motivation in mind, the proposal then defines the following goals:

- Bring OpenGL to a level that reflects the capabilities and performance of current and near-future graphics hardware.
- Provide a vision for the development of future generations of programmable graphics hardware.
- Reduce the need for existing and future extensions to OpenGL by replacing complexity with programmability.

The OpenGL 2.0 specification is currently being developed with broad support from industry and OEMs. The scheduled first draft of the specification is scheduled for June 2002, with the first public review draft at SIGGRAPH 2002 in July. Initial implementations of OpenGL 2.0 should occur around December 2002, and full implementations by July 2003.

6 Conclusion

It has been shown that programmable shading is a viable technique for scientific visualization. Programmable shading will allow for the creation of new techniques for the examination of data, and provide alternative ways of implementing older techniques. With the advent of hardware accelerated programmable shading, and the APIs necessary to support it, this technique will not only become a standard part of scientific visualization, but will change how visualization is done.

At the ARL MSRC, we are beginning to apply these techniques to the data generated by DoD researchers. It is expected that these techniques and shaders will be provided to the DoD user community over the next several years.

7 References

- Akeley, K., Hanrahan, P. “Real Time Graphics Architectures”, CS448A - Topics in Computer Graphics, Stanford University, 2001, <http://graphics.stanford.edu/courses/cs448a-01-fall/>
- Appel, A. “Some Techniques for Shading Machine Renderings of Solids”, In *Proceedings of the AFIPS Spring Joint Computer Conference*, AFIPS Press, Vol. 32, 1968, 37-49
- ART 2002. Advanced Rendering Technology product web site, <http://www.art-render.com>
- ATI 2001. ATI Radeon 8500 product web site, <http://www.ati.com/products/pc/radeon8500128/index.html>
- Blinn, J.F., Newell, M.E. “Texture and Reflection in Computer Generated Images”, In *Communications of the ACM*, 19, 10 (Oct. 1976), 542-547
- Blinn, J.F. “Models of Light Reflection for Computer Synthesized Pictures”, In *Proceedings of ACM SIGGRAPH 1977*, 192-198
- Blinn, J.F. “Simulation of Wrinkled Surfaces”, In *Proceedings of ACM SIGGRAPH 1978*, 286-292
- Bock, D. Visualization Shading Project web site, <http://woodall.ncsa.uiuc.edu/dbock/projects/VisShade/>
- Bouknight, J.W. “A Procedure for Generation of Three-Dimensional Half-Toned Computer Graphics Presentations”, In *Communications of the ACM*, 13, 9 (Sept. 1970), 527-536
- Bouknight, J.W., Kelley, J.C. “An Algorithm for Producing Half-Toned Computer Graphics Presentation with Shadows and Movable Light Sources”, In *Proceedings of the AFIPS Spring Joint Computer Conference*, AFIPS Press, Vol. 36, 1970, 1-10
- Bui-Tuong, P. “Illumination for Computer Generated Pictures”, In *Communications of the ACM*, 18, 6 (June 1975), 311-317
- Cabral, B., Leedom, L. “Imaging Vector Fields Using Line Integral Convolution”, In *Proc. of ACM SIGGRAPH 1993*, 263-272.
- Catmull, E.E. “A Subdivision Algorithm for Computer Display of Curved Surfaces”, Ph. D. Dissertation, University of Utah, Salt Lake City, Utah, 1974
- Cook, R.L. “Shade Trees”, In *Proceedings of ACM SIGGRAPH 1984*, 223-231
- Cook, R.L., Carpenter, L., Catmull, E. “The REYES Image Rendering Architecture”, In *Proceedings of ACM SIGGRAPH 1987*, 95-102
- DirectX 2001, DirectX Home Page, <http://www.microsoft.com/windows/directx/default.asp>
- DOTC 2001. RenderMan-Compliant Renderers: Past, Present, and Future, <http://www.dotcsw.com/links.html>
- Flarg 2001. “Shrek Rendering Statistics”, http://www.flarg.com/shrek_stats.html
- Forssell, L., “Visualizing Flow Over Curvilinear Grid Surfaces Using Line Integral Convolution”, In *Proceedings of IEEE Visualization '94*
- Gouraud, H. “Computer Display of Curved Surfaces”, In *IEEE Transactions on Computers*, C-20, 6, 1971, 623-629
- Hanrahan, P., Lawson, J. “A Language for Shading and Lighting Calculations”, In *Proceedings of ACM SIGGRAPH 1990*, 289-298
- Interrante, V. “Illustrating Surface Shape in Volume Data via Principal Direction-Driven 3D Line Integral Convolution”, In *Proceedings of ACM SIGGRAPH 1997*, 109-116

- Kay, D.S., Greenberg, D. "Transparency for Computer Synthesized Images", In *Proceedings of ACM SIGGRAPH 1979*, 158-164
- NVIDIA 2002. GeForce4 Ti Product overview, http://www.nvidia.com/docs/lo/1467/SUPP/PO_GF4Ti_2.05.02.pdf
- OpenGL2 2002. "OpenGL 2.0 Overview", <http://www.3dlabs.com/support/developer/ogl2/whitepapers/index.htm>
- Peercy, M. S., Olano, M., Airey, J., Ungar, P. J., "Interactive Multi-Pass Programmable Shading", In *Proceedings of ACM SIGGRAPH 2000*, 425-432
- Pixar 2000. "The RenderMan Interface Specification", http://www.pixar.com/renderman/developers_corner/rispec/index.html
- Purcell, T.J., Buck, I., Mark, W.R., Hanrahan, P. "Ray Tracing on Programmable Graphics Hardware", to appear in *Proceedings of ACM SIGGRAPH 2002*
- Rheingans, P. "Opacity-modulating Triangular Textures for Irregular Surfaces", IEEE Visualization '96, pp. 219-225.
- SGI 2000. OpenGL Shader product web site, <http://www.sgi.com/software/shader>
- Stanford 2000. "Stanford Real-Time Programmable Shading Project", <http://graphics.stanford.edu/projects/shading>
- Sutherland, I.E., R. F. Sproull, and R. A. Schumacker, "A Characterization of Ten Hidden-Surface Algorithms", *Computing Surveys*, 6, 1, 1974, 1-55
- Watkins, G. "A Real Time Hidden Surface Algorithm", Ph. D. Dissertation, University of Utah, Salt Lake City, Utah, 1970
- Whitted, T. "An Improved Illumination Model for Shaded Display", In *Communications of the ACM*, 23, 6 (June 1980), 343-349
- Wylie, C., Romney, G.W., Evans, D.C., Erdahl, A.C. "Halftone Perspective Drawings by Computer", In *Proceedings of the Fall Joint Computer Conference*, 1967, Thomson Books, 49-58